SCA Developer's Guide

APPENDIX D

XML for a Sample Waveform

03/28/2002

Introduction

This appendix contains XML files for the simple XYZ waveform that was developed in sections 6.1.1 - 6.3.5 of the SCA Developer's Guide. Each profile contained in this document is preceded by a brief discussion explaining what it contains and how its contents are organized. The explanations of this appendix assume familiarity with the contents of Appendix A, which explains some of the basic concepts of XML.

Appendix D to the SCA 2.2 Specification provides a complete specification of capabilities that can be used in XML files for SCA-compliant applications and devices. The example XML files presented here use only a few of the capabilities available under SCA 2.2. The commentary in this appendix applies only to the options demonstrated in the example XML. No attempt is made to cover the various other options available, but using the other capabilities does follow the same patterns demonstrated in these examples.

Line numbers are provided for each XML file listed in this appendix. Line numbers are not part of XML structure, but they are included in this appendix as a means of readily identifying and/or locating the various components of the file.

Properties File - app_xyz.prf.xml

A properties file describes component attributes.

The first two lines of this file are the prolog, which is explained on page A-6 of Appendix A. DTD file ***properties.2.2.dtd*** embodies the Properties Descriptor requirements (specified in section D.4 of Appendix D to the SCA 2.2 Specification).

Lines 4 and 85 provide the start tag and corresponding end tag that enclose the root element for the XML file. The following child elements are contained within the root element:

?? **description** {line 6} is a character string of information about the file

?? A "**simple**" element is one way of defining a component attribute[1].
app_xyz.prf.xml contains six simple elements {lines 8-20, define frequency, lines 22-30 define transmit mode, lines 32-45 define noise squelch, lines 47-58 define AGC Squelch, lines 60-70 define data speed, and lines 72-83 define power level}

Within the simple start tag, certain element attributes may be defined in any order (and, in fact, two different orderings are used in **app_xyz.prf.xml**)

* **id** is the formal identifier for the component attribute, and is used within id-value pairs (such as are used by configure and query) to identify the attribute

* **name** may be used, by a GUI for example, to identify the component attribute in a more user-friendly manner

* **type** denotes how the value is handled by the software

* **mode** may be "readonly", "writeonly", or "readwrite".

A "simple" element may have child elements (note that different combinations of child elements are used for the various XYZ attributes):

* **description** is a character string of information about the component attribute

* **value** provides an initial value for the component attribute

* **units** denotes how the component attribute's value is to be interpretted

* **range** provides lower and upper values for the component attribute

* **kind** denotes how the component attribute is used; "configure" is the default value

[1] If it is not "simple", an attribute is "simplesequence", "test", "struct", or "structsequence", but the XYZ waveform happens to use only "simple".

```
<?xml version="1.0" ?>
<!DOCTYPE properties SYSTEM “properties.2.2.dtd”>

<properties>

    <description>This prf describes the properties associated with the XYZ application</description>

    <simple
        id="CONFIG_ULONG"
        type="ulong"
        name="frequency"
        mode="readwrite">
        <description>This property indicates the waveform frequency</description>
        <value>225000000</value>
        <units>Hz</units>
        <range
            min="225000000"
            max="399999975" />
        <kind kindtype="configure" />
    </simple>

    <simple
        id="CONFIG_BOOLEAN"
        type="boolean"
        name="transmit mode"
        mode="readwrite">
        <description>This property indicates whether or not transmission is enabled.
                    A value of TRUE (1) indicates transmission is enabled</description>
        <value>1</value>
    </simple>

    <simple
        id="CONFIG_USHORT"
        type="ushort"
        name="noise squelch"
        mode="readwrite">
        <description>This value holds the last programmed squelch value. Zero means squelch is off.
                    For digital, a non-zero value means squelch is enabled.
                    For analog, the value indicates the squelch level.</description>
        <value>10</value>
        <units>%</units>
        <range
            min="0"
            max="100" />
    </simple>

    <simple
        mode="readwrite"
        id="AGC_Squelch"
        type="long"
        name="AGC Squelch">
        <description>This property indicates the AGC Squelch for the modem</description>
        <value>-100</value>
```

```
            <units>dbm</units>
            <range
                min="-100"
                max="0" />
        </simple>

        <simple
            mode="readwrite"
            id="txn_speed"
            type="long"
            name="data speed">
            <description>transmit data speed</description>
            <value>2400</value>
            <range
                min="110"
                max="1000000" />
        </simple>

        <simple
            mode="readwrite"
            id="power_level"
            type="long"
            name="power level">
            <description>This is the modem's power level</description>
            <value>-30</value>
            <units>db</units>
            <range
                min="-50"
                max="10" />
        </simple>

    </properties>
```

Software Assembly Descriptor - app_xyz.sad.xml

The software assembly descriptor describes an application; it specifies which components comprise the application and how those components are interconnected.

The first two lines of this file are the prolog, which is explained on page A-6 of Appendix A. DTD file ***softwareassembly.2.2.dtd*** embodies the Software Assembly Descriptor requirements (specified in section D.6 of Appendix D to the SCA 2.2 Specification).

Lines 4 and 84 provide the start tag and corresponding end tag that enclose the root element for the XML file. The ***id*** element attribute {line 5} is a DCE UUID that uniquely identifies the application; this ***id*** is used elsewhere in the system to locate this particular application. The ***name*** element attribute {line 6} is a user-friendly means of identifying the application.

The following child elements are contained within the root element:

?? **description** {line 8} is a character string information about the file

?? **componentfiles** {lines 10-17} describes the various components which comprise the application. There is a **componentfile** element for each component. Within **componentfile**, the **localfile** element specifies the Software Package Descriptor that describes the component, while the ***id*** element attribute assigns a logical name that will be used to reference the component within the XML for this application. In the case of our XML example, lines 12-15 describe the XYZ component, and line 14 refers to the SPD file that is presented in pages D-9 through D-12 of this appendix.

?? **partitioning** {lines 19-26} describes how components may be loaded on the system - a **componentplacement** element allows any valid placement, while a **hostcollocation** element forces components to be on the same processor. The example XML refers to just one component, so **hostcollocation** is not used. Lines 21-24 describe the XYZ component. The ***componentfileref refid*** refers back to the value used in the ***componentfile id*** attribute in the **componentfile** element {line 22 refers back to line 13}, and ***componentinstantiation id*** is a DCE UUID that provides a unique reference to the component that can be used elsewhere in the XML files.

?? **assemblycontroller** {lines 28-30} identifies which component serves as the Assembly Controller[2]. The ***componentinstantiationref refid*** is the DCE UUID that was introduced as the ***componentinstantiation id*** in the **componentplacement** element {line 29 refers back to line 23}.

[2] see sections 6.2.5 and 6.3.5 of the SCA Developer's Guide for a discussion of the Assembly Controller

?? **connections** {lines 32-82} has a **connectinterface** child element for each connection. The XYZ example has three connections[3], so the XML has three **connectinterface** elements - lines 34-48 (XYZ-to-Audio connection), lines 50-64 (Audio-to-XYZ connection), and lines 66-80 (XYZ-to-Log connection)

Each **connectinterface** has two child elements - following the usual SCA API terminology, the element that corresponds to the originator of messages is called "**usesport**", and the element that corresponds to the receiver of messages is called "**providesport**".

Each port element has two child elements - a port identifier and a means of locating the component that contains the port. The port identifier (**usesidentifier** or **providesidentifier**) contains the string that its passed in the *getport()* call to identify which port is wanted. The SCA provides several means to locate the appropriate component; three of these means are used in **app_xyz.sad.xml**:

1. **componentinstantiationref** locates a component that was instantiated by the application; the DCE UUID that was introduced as the ***componentinstantiation id*** in the **partitioning** element picks a particular application{lines 38, 61, and 70 refer back to line 23}.

2. **deviceusedbythiscomponentref** locates a device that is known by the domain manager. In **app_xyz.sad.xml**, lines 44-45 and 55-56 use lines 22-23 to refer to lines 13-14 to refer to line 40 in **app_xyz.spd.xml**

3. **domainfinder** locates a service that is known by the domain manager. In **app_xyz.sad.xml**, line 76 locates any log service known to the domain manager; the **name** attribute could have been used to select a particular log service.

[3] Figure 6.1-1 of the SCA Developer's Guide shows six ports for the XYZ software. However, Port 4 was merely a convenient way of representing the connection to the modem hardware, and we assume that the UI will manage its own connection (Ports 0 and 1) since the usual procedure is to start the UI after constructing the application, including all connections, is finished.

```
<?xml version="1.0" ?>
<!DOCTYPE softwareassembly SYSTEM "softwareassembly.2.2.dtd">

<softwareassembly
      id="DCE:583B8E91-8F33-11D4-816E-00508B6A52E6"
      name="XYZandAudio">

      <description>This SAD is for example XYZ application attached to an audio device</description>

      <componentfiles>

            <componentfile
                  id="xyz_component">
                  <localfile name="app_xyz.spd.xml" />
            </componentfile>

      </componentfiles>

      <partitioning>

            <componentplacement>
                  <componentfileref refid="xyz_component" />
                  <componentinstantiation id="DCE:37C14D83-3C13-11d6-9E76-00104B373911" />
            </componentplacement>

      </partitioning>

      <assemblycontroller>
            <componentinstantiationref refid=" DCE:37C14D83-3C13-11d6-9E76-00104B373911" />
      </assemblycontroller>

      <connections>

            <connectinterface>

                  <usesport>
                        <usesidentifier>modem_out_port</usesidentifier>
                        <componentinstantiationref refid=" DCE:37C14D83-3C13-11d6-9E76-00104B373911" />
                  </usesport>

                  <providesport>
                        <providesidentifier>audio_in_port</providesidentifier>
                        <deviceusedbythiscomponentref
                              refid=" DCE:37C14D83-3C13-11d6-9E76-00104B373911"
                              usesrefid="AudioDevice1" />
                  </providesport>

            </connectinterface>

            <connectinterface>

                  <usesport>
```

```
                        <usesidentifier>audio_out_port</usesidentifier>
                        <deviceusedbythiscomponentref
                              refid=" DCE:37C14D83-3C13-11d6 -9E76-00104B373911"
                              usesrefid="AudioDevice1" />
                  </usesport>

                  <providesport>
                        <providesidentifier>modem_in_port</providesidentifier>
                        <componentinstantiationref refid=" DCE:37C14D83-3C13-11d6 -9E76-00104B373911" />
                  </providesport>

            </connectinterface>

            <connectinterface>

                  <usesport>
                        <usesidentifier>log_out_port</usesidentifier>
                        <componentinstantiationref refid=" DCE:37C14D83-3C13-11d6 -9E76-00104B373911" />
                  </usesport>

                  <providesport>
                        <providesidentifier>log_in_port</providesidentifier>
                        <findby>
                              <domainfinder type="log" />
                        </findby>
                  </providesport>

            </connectinterface>

      </connections>

</softwareassembly>
```

Software Component Descriptor for Application - app_xyz.scd.xml

The software component descriptor describes the interfaces associated with an SCA-compliant component.

Each interface is uniquely identified by using a CORBA respository identifier, which consists of three parts: the prefix "IDL", a scoped type name, and a version number. Thus, for example, the CF::Resource interface defined by version 2.2 of the SCA is identified as "IDL:CF/Resource:2.2".

The first two lines of this file are the prolog, which is explained on page A-6 of Appendix A. DTD file ***softwarecomponent.2.2.dtd*** embodies the Software Component Descriptor requirements (specified in section D.5 of Appendix D to the SCA 2.2 Specification).

Lines 4 and 70 provide the start tag and corresponding end tag that enclose the root element for the XML file. The following child elements are contained within the root element:

?? **corbaversion** {line 6} indicates the version of CORBA that the delivered component suppports.

?? **componentrepid** {line 8} identifies which interface the component implements. A device must implement CF::Device, an application must implement CF::Resource, and an application factory must implement CF::ResourceFactory. Thus, the ***repid*** for this element always identifies CF::Device, CF::Resource, CF::ResourceFactory, or an interface which inherits one of these interfaces, even though the component may implement other interfaces also[4].

?? **componenttype** {line 10} for the example XYZ waveform is "resource". Other valid values are "device", "resourcefactory", "domainmanager", "log", "filesystem", "filemanager", "devicemanager", "eventservice", and "namingservice".

?? **componentfeatures** {lines 12-48} } has a child element named **ports** {lines 14-46}, which has a "**provides**" child element for each provides port and a "**uses**" child element for each uses port.

Each **provides** port {lines 16-20, 22-26, and 28-32} has two element attributes and one optional child element

repid is the name that uniquely identifies the interface realized at that port (it should match the ***repid*** of the corresponding **interfaces** element - see lines 50-68)

[4] In fact, the example XYZ interface implements three interfaces: CF::Resource, XYZ::Control_Physical_XYZ, and XYZ::UserProvider_Physical_XYZ.

providesname is the port identifier (it should match some **providesidentifier** child element of a **connectinterface** element in the SAD file).

Optional child element **porttype** can be "control", "data", "responses" or "test".

Each **uses** port {lines 34-38 and 40-44} has two element attributes and one optional child element

repid is the name that uniquely identifies the interface realized at that port (it should match the ***repid*** of the corresponding **interfaces** element - see lines 50-68)

usesname is the port identifier (it should match some **usesidentifier** child element of a **connectinterface** element in the SAD file).

Optional child element **porttype** can be "data", "control", "responses" or "test".

For example, the ***repid*** on line 41 matches the ***repid*** on line 57, and "log_out_port" appears as the ***usesname*** on line 42 of this file and as the **usesidentifier** on line 69 of **app_xyz.sad.xml**.

For both **provides** and **uses** ports, **porttype** defaults to "control" if the element is omitted. In general, **porttype** corresponds to interface types specified by the API Supplement[5] - "control" corresponds to an API "B" (non-real-time) interface and "data" corresponds to an API "A" (real-time) interface.

?? **interfaces** {lines 50-68} has an **interface** child element for each interface that the component provides, uses, or supports. ***repid*** is the name that uniquely identifies the interface.
name provides a character-based non-qualified (i.e., "user-friendly") way of identifying the interface.

[5] see Section 3 of the SCA Developer's Guide

```
<?xml version="1.0" ?>
<!DOCTYPE softwarecomponent SYSTEM “softwarecomponent.2.2.dtd”>

<softwarecomponent>

    <corbaversion>2.2</corbaversion>

    <componentrepid repid="IDL:CF/Resource:2.2" />

    <componenttype>resource</componenttype>

    <componentfeatures>

        <ports>

            <provides
                repid="IDL:CF/Resource:2.2"
                providesname="resource_in_port" />
                <porttype type="control"/>
            </provides>

            <provides
                repid="IDL:XYZ/Control_Physical_XYZ:2.2"
                providesname="control_in_port" />
                <porttype type="control"/>
            </provides>

            <provides
                repid="IDL:XYZ/UserProvider_Physical_XYZ:2.2"
                providesname="modem_in_port" />
                <porttype type="data"/>
            </provides>

            <uses
                repid="IDL:XYZ/UserProvider_Physical_XYZ:2.2"
                usesname="modem_out_port" />
                <porttype type="data"/>
            </uses>

            <uses
                repid="IDL:LogService/Log:2.2"
                usesname="log_out_port" />
                <porttype type="data"/>
            </uses>

        </ports>

    </componentfeatures>

    <interfaces>

        <interface
            repid="IDL:CF/Resource:2.2"
            name="**resource**" />

        <interface
            repid="IDL:LogService/Log:2.2"
            name="log" />

        <interface
            repid="IDL:XYZ/Control_Physical_XYZ:2.2"
            name="control_physical_xyz" />

        <interface
            repid="IDL:XYZ/UserProvider_Physical_XYZ:2.2"
            name="userprovider_physical_xyz" />

    </interfaces>

</softwarecomponent>
```

Software Package Descriptor for Application - app_xyz.spd.xml

This software package descriptor is used to load an SCA-compliant component.

The first two lines of this file are the prolog, which is explained on page A-6 of Appendix A. DTD file ***softpkg.2.2.dtd*** embodies the Software Package Descriptor requirements (specified in section D.2 of Appendix D to the SCA 2.2 Specification).

Lines 4 and 47 provide the start tag and corresponding end tag that enclose the root element for the XML file. The ***id*** element attribute {line 5} is a DCE UUID that uniquely identifies the component, and is used elsewhere in the system to locate this particular component. The ***name*** element attribute {line 6} is a user-friendly means of identifying the component, and the ***version*** element attribute {line 7} identifies which variant of the component is being used.

The following child elements are contained within the root element:

?? **author** {lines 9-13} is used to provide information about the person(s) who wrote the XML; no requirements limit what can be entered in the child elements of **author**.

?? **descriptor** {lines 15-17} specifies which Software Component Descriptor provides interface information for the component being loaded.

?? **implementation** {lines19-37}describes a particular implementation of the component. The ***id*** element attribute {line 20} is a DCE UUID that uniquely identifies the particular implementation of the component, and is used elsewhere in the system to locate this implementation.

The **propertyfile** child element {lines 22-25} specifies which particular property file should be used with this implementation; line 24 specifies that the file presented on pages D-2 and D-3 of this appendix should be used with the implementation described in this file.

The **code** child element {lines 27-31} describes the instructions that provide the functionality of this implementation. The ***type*** element attribute {line 28} can be "Executable" (a main process to be loaded and executed), "KernelModule" (load only), "SharedLibrary" (dynamically linked) or "Driver" (load only). Child element **loadfile** {line 29} specifies the file containing the actual code, and **entrypoint** {line 30} specifies where execution should begin (if that information is needed in the context in which the code will be used[6]).

[6] For example, if the operating system is LynxOS, the entrypoint is understood to be 'main', so this information need not be provided in an SPD that will be used with LynxOS.

The **os** child element {lines 33-35} is interpretted as dependency information and is compared against appropriate allocation properties.

?? **usesdevice** {lines 39-45}describes any[7] relationship between the component and a device in the system.

Child element **id** {line 40} provides the label that will be used to locate this device from any other XML in the assembly; for example, line 40 provides a label that is referenced from line 45 of **app_xyz.sad.xml**. Child element **type** {line 41} allows the XML author to convey information to others who will examine the XML.

The **propertyref** child element provides information needed to locate the actual device, as ***refid*** {line 43} is the DCE UUID used in the device's SPD to identify a simple allocation property, and ***value*** provides information needed by the dependency checking.

[7] The word "uses" in this context has no connection to the use of "uses" as the alternative to "provides"

```
<?xml version="1.0" ?>
<!DOCTYPE softpkg SYSTEM "softpkg.2.2.dtd">

<softpkg
    id="DCE:4BD22011-69FD-11D4-816A-00508B6A52E6"
    name="xyz_component"
    version="revB">

    <author>
        <name>A&R</name>
        <company>Raytheon Company</company>
        <webpage>http://www.raytheon.com</webpage>
    </author>

    <descriptor>
        <localfile name="app_xyz.scd.xml" />
    </descriptor>

    <implementation
        id="DCE:4BD22012-69FD-11D4-816A-00508B6A52E6">

        <propertyfile
            type="property">
            <localfile name="app_xyz.prf.xml" />
        </propertyfile>

        <code
            type="Executable">
            <localfile name="xyz.out" />
            <entrypoint>xyzMain</entrypoint>
        </code>

        <os
            name="VxWorks"
            version="4.0" />

    </implementation>

    <usesdevice
        id="AudioDevice1"
        type="AUDIO_DEVICE">
        <propertyref
            refid="B4839010-3DA9-11d6-9E76-00104B373911"
            value="1" />
    </usesdevice>

</softpkg>
```